Introducció a OpenCV i Qt

Lluís Garrido i Àlex Pardo

15 d'Abril de 2015

Resum

Aquesta pràctica té per objectiu introduir-vos amb el programari que s'utilitzarà al llarg de les pràctiques de Software Concurrent: les llibreries OpenCV i Qt.

Índex

1 OpenCV

OpenCV (Open Source Computer Vision) és una llibreria especialitzada en visió per ordinador. Va ser dissenyada per a tenir una gran eficiència computacional i per a aplicacions en temps real. Empreses com Google, Yahoo, Intel, IBM, Sony o Honda utilitzen activament aquesta llibreria per desenvolupar aplicacions de visió per ordinador.

La llibreria disposa de més de 2500 algorismes incloent tant algorismes clàssics com els de l'estat de l'art. Aquests algorismes poden ser utilitzats per a detectar i reconèixer cares, identificar objectes, classificar accions humanes a vídeos, seguir els moviments d'una càmera, etc.

Aquesta llibreria està desenvolupada en llenguatge C++ i disposa d'interfícies per als llenguatges C, C++, Python i Java. Està suportada als sistemes operatius Windows, Linux, Android i Mac OS.

En el context de l'assignatura de Software Concurrent s'utilitzarà aquesta llibreria per implementar la funcionalitat i els algorismes de visió per ordinador de l'aplicació que es desenvoluparà. A continuació es mostren alguns exemples d'ús d'aquesta llibreria. Els tres exemples que es mostren a continuació són:

1. Lectura i visualització d'una imatge.
2. Càlcul d'un histograma d'una imatge.
3. Cerca d'una cara a una imatge.

Aquests exemples han estat obtinguts del tutorial de OpenCV, veure l'enllaç http://docs.opencv.org/doc/tutorials/tutorials.html, i tenen per objectiu mostrar la capacitat de la llibreria OpenCV. Es demana a l'estudiant compilar i executar aquests exemples per assegurar una bona comprensió del funcionament d'aquesta llibreria.

1.1 Lectura d'imatges

A la Figura 1 es mostra un exemple per a la lectura d'una imatge fent servir la llibreria OpenCV.

Per compilar el codi es poden utilitzar diversos mètodes. Per exemple, es pot utilitzar una interfície de desenvolupament integrada com Eclipse o Qt-Creator per fer-ho. Això implica configurar els camins (*paths*) de les llibreries

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>

using namespace cv;
using namespace std;

int main( int argc, char** argv )
{
  if( argc != 2)
  {
    cout <<" Usage: read image" << endl;
    return -1;
  }

  Mat image;
  image = imread(argv[1], CV_LOAD_IMAGE_COLOR);

  if (!image.data)
  {
    cout <<  "Could not open or find the image" << std::endl ;
    return -1;
  }

  namedWindow( "Read Demo", CV_WINDOW_AUTOSIZE );
  imshow( "Read Demo", image );

  waitKey(0);
  return 0;
}
```

Figura 1: Codi d'exemple per a la lectura d'imatges a OpenCV. La imatge llegida es mostra per pantalla.

d'OpenCV a l'hora de definir el projecte. En aquest cas, però, utilitzarem un conjunt d'eines que ens ofereix la línia de comandes. L'estudiant és lliure d'utilitzar un altre mètode si així ho voleu.

Cal seguir els següents passos per compilar el codi:

1. Obriu un terminal i col·loqueu-vos al directori on hi hagi el codi font de la Figura 1. Suposarem que el codi font s'ha emmagatzemat amb el nom `read.cpp`.

2. Genereu un fitxer `CMakeLists.txt` al mateix directori en què es trobi el codi font `read.cpp` i que contingui el text indicat a continuació

   ```
   PROJECT( opencv )
   FIND_PACKAGE( OpenCV REQUIRED )
   ADD_EXECUTABLE( read read.cpp )
   TARGET_LINK_LIBRARIES( read ${OpenCV_LIBS} )
   ```

3. Executeu la següent instrucció en el directori en què es trobi `read.cpp`.

   ```
   $ cmake .
   ```

4. La instrucció `cmake` llegeix el fitxer `CMakeLists.txt` i genera els corresponents fitxers `Makefile`. Veureu que per pantalla apareixen una sèrie de missatges indicant què està fent `cmake`.

5. Un cop generat els fitxers `Makefile` podem compilar l'aplicació executant

   ```
   $ make
   ```

6. En compilar es mostrarà per pantalla l'estat de la compilació. Si no hi ha hagut errors de compilació s'indicarà per pantalla que s'ha generat l'executable `read`.

7. Finalment, podem executar l'aplicació amb

   ```
   $ ./read
   ```

Observeu que aquesta aplicació permet visualitzar la imatge a pantalla. Aquesta funcionalitat està suportada per la llibreria OpenCV a través de Qt, tot i que les possibilitats que ens ofereix OpenCV per tal de generar interfícies gràfiques d'usuari és molt limitada. És per això que, en el context de l'assignatura de Software Concurrent, s'utilitzarà la llibreria Qt per a la implementació de les interfícies gràfiques d'usuari.

1.2 Histograma

A la Figura 2 es mostra un exemple en què es calcula l'histograma d'una imatge en color. Només s'hi mostra aquella part del codi que llegeix la imatge i en calcula l'histograma. El codi complet de l'exemple es troba amb l'enunciat de la pràctica.

Per compilar aquest codi utilitzarem el mateix mètode que abans: un fitxer `CMakeLists.txt`. En aquest cas el contingut del fitxer ha de ser el que s'indica a continuació

```
PROJECT( opencv )
FIND_PACKAGE( OpenCV REQUIRED )
ADD_EXECUTABLE( histogram histogram.cpp )
TARGET_LINK_LIBRARIES( histogram ${OpenCV_LIBS} )
```

Un cop compilat el codi el podem executar amb

```
$ ./histogram
```

En cas que li passem per argument una imatge veurem que es visualitzen per pantalla els histogrames de les components R, G i B de la imatge.

1.3 Detecció de cares

Finalment mostrem un exemple d'una aplicació que detecta cares a una imatge. El codi per a detectar una cara a una imatge i marcar-la és el que es mostra a la Figura 3. El codi complet de l'exemple es pot trobar juntament amb aquesta pràctica. Com es pot veure, el codi en OpenCV per utilitzar aquesta funcionalitat és força curta.

Proveu de generar el fitxer CMakeLists.txt per compilar el codi. Un cop ho hagueu fet executeu el codi amb diverses imatges que continguin cares. En particular, s'inclouen diverses imatges amb aquesta pràctica que podeu fer servir d'exemple. Observeu que l'algorisme no és perfecte i no pot detectar totes les cares a les imatges.

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
#include <stdio.h>

using namespace std;
using namespace cv;

int main( int, char** argv )
{
  Mat src, dst;

  if( argc != 2)
  {
    cout <<" Usage: histogram image" << endl;
    return -1;
  }

  /// Load image
  src = imread( argv[1], 1 );
  if( !src.data ) { return -1; }

  /// Separate the image in 3 places ( B, G and R )
  vector<Mat> bgr_planes;
  split( src, bgr_planes );

  /// Establish the number of bins
  int histSize = 256;

  /// Set the ranges ( for B,G,R) )
  float range[] = { 0, 256 } ;
  const float* histRange = { range };

  bool uniform = true; bool accumulate = false;

  Mat b_hist, g_hist, r_hist;

  /// Compute the histograms:
  calcHist( &bgr_planes[0], 1, 0, Mat(), b_hist, 1, &histSize,
            &histRange, uniform, accumulate );
  calcHist( &bgr_planes[1], 1, 0, Mat(), g_hist, 1, &histSize,
            &histRange, uniform, accumulate );
  calcHist( &bgr_planes[2], 1, 0, Mat(), r_hist, 1, &histSize,
            &histRange, uniform, accumulate );

  ...
}
```

Figura 2: Codi que llegeix una imatge, en calcula l'histograma i el mostra per pantalla. El codi complet de l'exemple es troba amb l'enunciat de la pràctica.

```
using namespace std;
using namespace cv;

void detectAndDisplay(Mat frame)
{
  vector<Rect> faces;
  Mat frame_gray;

  CascadeClassifier face_cascade;

  // Load the cascades
  if (!face_cascade.load("haarcascade_frontalface_alt.xml")) {
    cout << "-- Error loading cascade\n" << endl; exit(1);
  }

  cvtColor(frame, frame_gray, CV_BGR2GRAY);
  equalizeHist(frame_gray, frame_gray);

  // Detect faces
  face_cascade.detectMultiScale( frame_gray, faces, 1.1, 2,
                                 0|CV_HAAR_SCALE_IMAGE, Size(30, 30));

  // Draw ellipses in image
  for( int i = 0; i < faces.size(); i++ )
  {
    Point center(faces[i].x + faces[i].width*0.5,
                 faces[i].y + faces[i].height*0.5 );
    ellipse(frame, center, Size(faces[i].width*0.5,
            faces[i].height*0.5), 0, 0, 360, Scalar(255, 0, 255 ),
            4, 8, 0);
  }
  // Show result
  imshow("Face detection demo", frame );
}
```

Figura 3: Funció que detecta les cares a la imatge d'entrada i les visualitza per pantalla. El codi complet de l'exemple es troba amb l'enunciat de la pràctica.

2 Llibreria Qt

Qt és la llibreria multiplataforma que s'utilitzarà al llarg de les pràctiques d'aquesta assignatura per desenvolupar la interfície gràfica d'usuari. Qt és un programari lliure i de codi obert i s'utilitza a l'entorn d'escriptori KDE i altres múltiples aplicacions com per exemple Google Earth, Opera o Skype.

Qt està programat de forma nativa en el llenguatge de programació C++ i utilitza també aquest llenguatge per a la seva API. Permet desenvolupar aplicacions d'interfície d'usuari amb molt poques línies de codi. Qt utilitza a més els recursos del sistema per a dibuixar les interfícies a pantalla, cosa que vol dir que l'aplicació semblarà nativa del sistema operatiu (és a dir, a Windows s'assemblarà a una aplicació de Windows mentre que a Mac s'assemblarà a una aplicació de Mac).

Per desenvolupar aplicacions Qt es poden utilitzar aplicacions de desenvolupament integrades com per exemple QtCreator. En els exemples que mostrarem a continuació, però, no cal utilitzar aquesta aplicació per compilar i executar les aplicacions.

De forma similar a la secció anterior, es mostren a continuació alguns exemples d'ús per a crear aplicacions senzilles. Per a més exemples de creació d'interfícies d'usuari es pot consultar aquest tutorial de a l'enllaç: http://zetcode.com/gui/qt4/

2.1 Visualització d'una imatge

A l'exemple de la Figura 4 es mostra un exemple bàsic que permet visualitzar una imatge a pantalla mitjançant les funcions de la llibreria Qt.

Per a compilar aquest codi utilitzarem el mateix mètode que hem utilitzat per OpenCV: un fitxer `CMakeLists.txt`. El fitxer `CMakeLists.txt` que permet generar els fitxers `Makefile` per compilar és el següent

```
PROJECT(example)
FIND_PACKAGE(Qt4 REQUIRED)
INCLUDE(${QT_USE_FILE})
ADD_EXECUTABLE(qimage qimage.cpp)
TARGET_LINK_LIBRARIES(qimage ${QT_LIBRARIES})
```

Observeu que els continguts del fitxer CMakeLists.txt és molt similar a l'utilitzat amb OpenCV. En aquest cas s'ha adaptat el fitxer perquè compili amb Qt.

```
#include <QtGui>

int main(int argc, char* argv[])
{
  QApplication app(argc, argv);

  QImage image("image.jpg");

  QLabel label;
  label.setPixmap(QPixmap::fromImage(image));
  label.show();

  return app.exec();
}
```

Figura 4: Codi exemple per mostrar una imatge a pantalla mitjançant Qt.

Per a generar els fitxers Makefile cal executar

```
$ cmake .
```

L'aplicació es pot compilar amb

```
$ make
```

Finalment, si la compilació s'ha realitzat correctament podem executar l'aplicació amb

```
$ ./qimage
```

2.2 Visualització d'una taula d'imatges

L'exemple de la Figura 5 mostra com visualitzar múltiples imatges en una finestra organitzant-los en forma de taula. Aquesta aplicació us serà útil per a les pràctiques de Software Concurrent.

En aquest cas el codi font està format per dos fitxers (tot i que es podria haver fet servir un únic fitxer). Observeu el continguts del fitxer `CMakeLists.txt` associat que hi ha a continuació i, en particular, com s'hi indiquen els fitxers a compilar.

```
PROJECT(example)
FIND_PACKAGE(Qt4 REQUIRED)
INCLUDE(${QT_USE_FILE})
ADD_EXECUTABLE(qtgrid main.cpp images.cpp)
TARGET_LINK_LIBRARIES(qtgrid ${QT_LIBRARIES})
```

```
#include <QPushButton>
#include <QGridLayout>
#include <QFileInfo>
#include <QLabel>
#include "images.h"

QGridImages::QGridImages(QWidget *parent)
    : QWidget(parent)
{
  QGridLayout *grid = new QGridLayout(this);

  QImage copy;
  QString tempFileName;
  QList<QImage> images;
  QList<QString> filesList;
  filesList << "image1.png" << "image2.png" << "image3.png";

  foreach(QFileInfo fileInfo, filesList)
  {
    tempFileName = fileInfo.fileName();
    QImage image(tempFileName);
    copy = image.scaled(200,200,Qt::KeepAspectRatio);
    images.append(copy);
  }

  for (int i = 0; i < 3; i++) {
    for(int j = 0; j < 3; j++) {
      QPixmap p(QPixmap::fromImage(images[i]));
      QLabel *label = new QLabel(this);
      label->setPixmap(p);
      grid->addWidget(label, i, j);
    }
  }

  setLayout(grid);
}
```

Figura 5: Constructor de la classe que permet visualitzar múltiples imatges organitzant-los en forma de taula. El codi complet de l'exemple es troba amb l'enunciat de la pràctica.

2.3 Ús conjunt d'OpenCV i Qt

En aquesta secció és mostra com utilitzar la llibreria OpenCV i Qt conjuntament. En particular, s'utilitza OpenCV per llegir una imatge i Qt per visualitzar-la. La pràctica que realitzareu al llarg d'aquestes setmanes utilitzarà OpenCV com a llibreria per manipular imatges, mentre que s'utilitzarà Qt per a la interfície d'usuari.

A la Figura 6 es mostra el codi d'exemple. Observar que es pot transformar de forma senzilla del tipus Mat d'OpenCV al tipus QImage de Qt.

3 Tasques a realitzar

Es comenten aquí les tasques a realitzar fins la propera setmana. Aquestes tasques tenen per objectiu consolidar els coneixements d'OpenCV i Qt.

1. Tot i que en aquesta assignatura no cal tenir un coneixement alt de programació de classes en C++, se us recomana donar un cop d'ull al tutorial que hi ha aquí . En particular, reviseu la secció de "Object Oriented Programming" així com els "Templates" dels "Advanced Concepts".

2. A les pràctiques s'utilitzarà l'histograma per poder cercar imatges en una base de dades. Llegiu-vos, per tant, la documentació associada a la funció `calcHist` d'OpenCV. Observeu que en aquest enllaç s'inclouen funcions que permeten realitzar la comparació entre dos histogrames.

3. Agafeu els exemples de les Seccions 2.2 i 2.3 i modifiqueu-los per tal d'afegir-hi aquelles eines que cregueu necessàries per a la vostra aplicació. Tal com s'ha comentat abans l'objectiu central es realitzar una aplicació que permeti cercar imatges. La base de dades serà un fitxer de text pla que contingui a cada línia, el nom del fitxer d'una imatge. Aleshores, aneu pensant en com afegir la següent funcionalitat (a nivell de interfície d'usuari) a la vostra aplicació:

   (a) Un botó per indexar les imatges de la base de dades. En seleccionar aquest botó l'aplicació demanarà el fitxer de text pla que conté les imatges de les quals es calcularà l'histograma.

```
#include <iostream>

// Qt
#include <QtGui>

// OpenCV
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>

using namespace std;
using namespace cv;

int main(int argc, char **argv)
{
  if( argc != 2)
  {
    cout <<" Usage: qtview image" << endl;
    return -1;
  }

  // Llegim les dades via OpenCV
  Mat img;
  img = imread(argv[1], CV_LOAD_IMAGE_COLOR);

  if(!img.data)
  {
    cout <<  "Could not open or find the image" << endl;
    return -1;
  }

  // Transformem a QImage de Qt
  QImage qimg = QImage((uchar*) img.data, img.cols, img.rows,
                       img.step, QImage::Format_RGB888);

  // Aplicació Qt que visualitza la imatge
  QApplication app(argc, argv);

  QLabel label;
  label.setPixmap(QPixmap::fromImage(qimg));
  label.show();

  return app.exec();
}
```

Figura 6: Exemple bàsic d'ús conjunt de la llibreria OpenCV i Qt. Mitjançant la llibreria OpenCV llegim una imatge i la visualitzem amb les eines de Qt.

(b) Un botó per poder seleccionar una imatge qualsevol que no sigui de la base de dades. Es buscaran les imatges similars a la base de dades mitjançant l'histograma.

(c) La visualització de les imatges més similars a la seleccionada mitjançant la plantilla de la Secció 2.2. Una forma senzilla de fer-ho és indicar una llista amb els índexs corresponents del fitxer de text pla. Així, per exemple, per a una determinada imatge només caldrà indicar que les imatges més similars s´on les imatges (per aquest ordre) 16, 3, 294, 3943 i 593.